Deep Learning Notes

Carnegie Mellon University 11-785 Introduction To Deep Learning

`https://deeplearning.cs.cmu.edu/F21/index.html`

The source Notes have fantastic visualisations and this document merely contains a few asides, thoughts, comments, and lowkey verification of some comprehension. As always, a skim is useful, as is gazing upon all of the provided code, and assignments task statements to get an idea about what this is all about.

We shall see about pushing updates to this document in public actually for the time being it may be strategic to keep my notes private so that a potential interviewer will not decide not to ask me something based upon it appearing in my notes... but certainly if one has indications about topics one ought to find interview tasks on those topics.

Neural Groupings

Different intensities of activation of A lead to the differences in when X and Y are activated.

Connectionism: Network of processing elements, all world knowledge is stored in the connections between the elements.

Turing's Connectionist Machines: A-type Machines, B-type Machines.

Parallel Distributed Processing: a set of processing units, a state of activation, an output function for each unit, a pattern of connectivity amongst units, a propagation rule for propagating patterns of activities through the network of connectivities, an activation rule for combining the inputs impinging on a unit with the current state of that unit to produce a new level of activation for the unit, a learning rule whereby patterns of connectivity are modified by experience, an environment within which the system must operate.

Hebb: A learning mechanism. When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased. As A repeatedly excites B, its ability to excite B improves. "Neurons that fire together wire together."

Hebbian Learning: if neuron X repeatedly triggers neuron y, the synaptic knob

connecting x to y gets larger. $w_{xy} = w_{xy} + \eta xy$.

Simplified Perceptron Model: number of inputs combine linearly, threshold logic, fire if combined input exceeds threshold, $Y = 1$ if $\sum w_i x_i - T \geq 0$ else 0.

The Universal Model: originally assumed could represent any Boolean circuit and perform any logic. Boolean tasks, learning algorithm $w = w + \eta(d(x) - y(x))x$, sequential learning, $d(x)$ is the desired output in response to input x, $y(x)$ is the actual output in response to x, update the weights whenever the perceptron output is wrong, update the weight by the product of the input and the error between the desired and actual outputs, proved convergence for linearly separable classes. Recall here that linearly separable really means there exists a hyperplane decision boundary which splits a point set as desired completely, and one perhaps could modify these statements with threshold functions for roughly linear separable under some metric.

Perceptron: easily shown to produce any Boolean gate except for XOR exclusive or which requires a networked multi layer structure of neurons. So this is known as the multi-layer perceptron also due to Minsky.

Multi Layer Perceptron: can compose arbitrarily complicated Boolean functions. Of course one supposes that this is immediately trivial from the previous note and this produces a simple construction with an actual quantitative upper bound on size.

Perceptron: notation can include visual with graph of output function in terms of input sum of inputs with constant term so here one can have without loss of generality it is if and only if the overall sum or function at the neuron is ≥ 0 and like $\theta(z) = 1$ if $z \geq 0$ else 0. And this can be represented $y = \theta(\sum w_i x_i + b)$. So this would be a classical canonical like linear to binary thresholded output neuron model. And another canonical one is like the output is real valued $y = \text{sigmoid}(\sum w_i x_i + b)$ sometimes viewed as the "probability" of firing.

Composing "Complicated Decision Boundaries": there exists a network of units with a single binary output that fires if and only if the input in $\mathbb{R}^2$ input coordinates of a simple n-gon. It is a standard break down a bunch of sub components if and only if on the correct side of a boundary line half plane. And then more complex decision boundary constructions with comments on the number of layers levels needed to produce such constructions.

Complex Decision Boundaries: classification problems, finding decision boundaries

in high-dimensional space, can be performed by a multi layer perceptron, can classify real valued inputs. Their example is the MNIST data set with a small image file to be classified into a digit and thus the input dimension is 784 but a small neural network implicitly represents and executes the classification task.

Recall one can mix up neuron types, different layers, different mixups mashups all sorts of functions can exist. And some simple models work out relatively well in the real world practice praxis of applied machine learning.

Multi Layer Perceptron Continuous Valued Regression: a simple 3 unit multi layer perceptron which is 1 if and only if the input lies between 2 thresholds i.e. $x \in [T_1, T_2]$. And then this extends taking the limit of the Riemann sum e.g. to any sort of a well it is for smoothly continuous polynomially approximable locally yadda and also joint multidimensional functions ought to be noted also supported with this sort of a mashup. Anything can be changed and produce some mathematical structure.

Another concrete example stressed is that multi layer perceptrons can represent probability distributions. They can model both posterior and prior distributions of data, and they can generate data from complicated, or even unknown distributions.

Neural Networks Examples Tasks: voice signal recording measurement to transcription, image to text caption, game state to next move. Still no real deep comments on computational complexity praxis.

Sigmoid Soft Logistic Perceptron: recall $z = \sum w_i x_i + b$ then output is $y = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-(\sum w_i x_i + b)}}$. Some other functions "activations" tanh or $\ln(1 + e^z) = \ln(1 + e^{\sum w_i x_i + b})$.

Linear Combination: $\sum w_i x_i$

Affine Function: $\sum w_i x_i + b$

Deep Structures: in any directed graph with input source nodes and output sink nodes, "depth" is the length of the longest path from a source to a sink. Could simultaneously breadth first search from the source nodes to produce an $O(E + V)$ algorithm instantly to compute "depth". Deep neural network structure may refer to depth > 2. The "depth" of a layer is with respect to the input layer. There exists a trivial 1 layer universal AND gate for example associate the weight of 1 to all the M dudes who must be true and -1 to all

dudes who must be false and then boom threshold sum of M if and only if the big AND is true. And there exists a trivial 1 layer universal OR gate for example same associations with threshold $M - N + 1$ will be true if and only if at least 1 Boolean input is as desired. And there exists a subset size majority clearing threshold with each weight 1 and a desired threshold sum.

How many layers are needed in a multi layer perception to produce a particular Boolean function?

Recall that a Boolean function is just its truth table. And for a truth table one can simply produce 1 layer which contains a node for each output 1 vector and make the appropriate connections with the inputs. Known as disjunctive normal form. So this way we can do it in 1 layer with that number of nodes and $O(mn)$ edges there with the relevant weights. This is called between the input and output a 1 hidden layer multi layer perceptron.

I wish I had been given this task as a puzzle! Rather than having it ruined by choosing the strategy of simply reading through these notes and chillaxing!

Reducing A Boolean Function: Karnaugh Map, adjacent boxes can be "grouped" to reduce the complexity of the particular representation disjunctive normal form. Find groups, express as reduced disjunctive normal form. So for example they show the locus of inputs which we want to evaluate to true 1. And then represent $O = \bar{Y}\bar{Z} + \bar{W}X\bar{Y} + \bar{X}Y\bar{Z}$.

Worst Case Construction Analysis: taking the input sum modulo 2 will require 2^{N-1} perceptrons in 1 hidden layer.

Deep!

So the sum of the input modulo 2 is of course the universal XOR exclusive or operation whence we can simply inductively construct off of the 3 unit 2 layer construction of Minsky and produce the target output with only $3(N-1)$ perceptrons. But depth grows $O(n)$! Inductive $O(\log(N))$ construction pairing terms with $O(N)$ units.

Number Of Parameters: this is the number that really matters in software or hardware implementations. Networks that require an exponential number of neurons will certainly require an exponential number of weights. The actual number of paramaters is like $O(V + E)$ is $O(E)$ so the number of connections and weights. Kind of hand waved assuming a certain precision float here would be

used to represent a weight this is all pretty hand waved and casual and there also of course exist other implementations in terms of real world hardware electrical engineering bio mimic whatever yadda can come into play here.

As an aside here note the definitions of what we might soon come into which are recurrent neural networks, and feedforward neural networks. So there exist useful structures where neurons' output cycles back as input.

More layers in a multi layer perceptron can mean less units, less edges weights parameters.

Sipser any Boolean parity circuit of depth d using AND, OR, NOT gates with unbounded input network structuring permitted must have size $2^{n^{\frac{1}{d}}}$.

Not all Boolean circuits have such clear depth versus size tradeoff.

Shannon: for $n > 2$, almost all n-input Boolean functions need more than $\frac{2^n}{n}$ Boolean gates regardless of depth.

Network Size: a multi layer perceptron is a universal Boolean function but can represent a given function only if it is sufficiently wide, deep, tradeoff, optimal width and depth depend on the number of variables and the complexity of the Boolean function, the minimal number of terms in the disjunctive normal form formula to represent it.

Even a network with a single hidden layer is a universal Boolean machine.

Can compose arbitrarily complex decision boundaries [ignoring some measure 0 stuff] with only 1 hidden layer to arbitrary precision by representing it as a union of sufficiently small disks. So multi layer perceptrons are sometimes called universal classifiers. Formal analyses typically view these as a category of arithmetic circuits. Compute polynomials over any field, Valiant et al. a polynomial of degree n requires a network of depth $\log^2(n)$. Depth/size analyses of arithmetic circuits is still a treasure trove of open research problems!

For the general case of N mutually intersecting hyperplanes in D dimensions, we will need $O\left(\frac{N^D}{(D-1)!}\right)$ weights assuming $N >> D$. Increasing input dimensions can increase the worst case size of the shallower network exponentially, but not the XOR net.

Story So Far: multi layer perceptrons with 1 hidden layer are universal Boolean machine, universal classifier, but may require exponentially large number of

perceptrons than a deep one, more expressive.

Perceptrons So Far: $z = \sum w_i x_i - T$ from inputs and output $y = f(z)$ is the "activation" function.

Alternate Type Of Neural Unit:
Radial Basis Functions: $z = ||x - w||^2$ from inputs and output $y = f(z)$. A typical activation function here would be output $y = f(z) = e^{-\beta z}$. Here β is a "bandwidth" parameter. Output is a function of the distance of the input from a "center" w is the parameter specifying the unit. But other similar activations may also be used, key aspect is radial symmetry, instead of linear symmetry. So think of in kernel regularisation or kernel whatever those various continuously smooth functions satisfy desiderata straight up can 0 out early on whatever.

Radial Basis Functions: networks are more effective approximators of continuous valued functions because a 1 hidden layer network only requires 1 unit per "cylinder".

How do we represent the input? How do we represent the output? How do we compose then network that performs the requisite function?

Redrawing The Neuron: now simply append a unit and edge for each unit with a component of 1 so that we may without loss of generality rewrite this bias term in as a simple linear combination extension $z = \sum w_i x_i$.

Feed Forward Network: ah this is what we have been working with thus far is the simple uni directional directed acyclic graph structure.

Learning The Network: want to learn, optimise the parameters weights values such that the given network architecture optimises towards producing the desired target output performance function.

Construct By Hand: so we have already done some of these for the construction of decision boundary classification tasks.

How to learn a network? So the whole point is that we do not know the function, if we have training data, e.g. set of images with class labels or speech recordings with transcriptions then we may use these as samples to produce a function network model.

The Perceptron Problem: with the modification addition the task isomorphs into computing the hyperplane through the origin which perfectly separates 2 point

sets. Finding the weights vector W such that the hyperplane described by $W^T X = 0$ perfectly separates the classes i.e. $W^T X$ is > 0 and < 0 for the sets.

Online Perceptron Solution: popular, perceptron algorithm, initialises W and incrementally updates it each time we encounter an instance that is incorrectly classified. Sounds like some modified intuitive smoothing windmilling type thingy.

Convergence Of Perceptron Algorithm: more to exposit how these sorts of argumentations can go down than anything else, this is not terribly technically tricky, but if the classes are linearly separable then convergence is guaranteed. After no more than $\left(\frac{R}{\gamma}\right)^2$ misclassifications. R is the length of the longest training points. γ is the best case closest distance of a training point from the classifier. Same as the margin in a Support Vector Machine. Intuitively, takes many increments of size γ to undo an error resulting from a step of size R.

So we go back to the 2 pentagon example really imagine 2 separate mutually exclusive convex hulls of point sets inside a larger ocean of a point set of the other class. How will we learn here? This will extend to the general case? Recall of course that a k-nearest neighbours approach may work alrightish here in some cases.

So they propose trying literally every possible setting subset of the blue dots such that we can learn a line that keeps all the red dots on 1 side is a terrible exponential algorithm which exists nevertheless to obtain perfect performance. It would immediately strike my intuition that, given this is in this course, and just intuition on humanity, this is probably canonical there probably exist very good algorithms for this task, it is probably in the well studied domain of this domain.

Least mean squares algorithms are a class of adaptive filter used to mimic a desired filter by finding the filter coefficients that relate to producing the least mean square of the error signal (difference between the desired and the actual signal). It is a stochastic gradient descent method in that the filter is only adapted based on the error at the current time.

Adaptive Linear Element: actually just a regular perceptron, differs in the learning rule.

During learning, minimise the squared error assuming z to be real output. The desired output is still binary!

Tasks, Examples, Interviews

Regular Wikipedia review.

Neural Networks

An Artificial Neural Network is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. An artificial neuron receives a signal then processes it and can signal neurons connected to it. The "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

Recurrent Neural Networks

A recurrent neural network is a class of artificial neural networks where connections between nodes form a directed or undirected graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Derived from feedforward neural networks, Recurrent Neural Networks can use their internal state (memory) to process variable length sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition. Recurrent neural networks are theoretically Turing complete and can run arbitrary programs to process arbitrary sequences of inputs.

The term "recurrent neural network" is used to refer to the class of networks with an infinite impulse response, whereas "convolutional neural network" refers to the class of finite impulse response. Both classes of networks exhibit temporal dynamic behavior. A finite impulse recurrent network is a directed acyclic graph that can be unrolled and replaced with a strictly feedforward neural network, while an infinite impulse recurrent network is a directed cyclic graph that can not be unrolled.

Both finite impulse and infinite impulse recurrent networks can have additional

stored states, and the storage can be under direct control by the neural network. The storage can also be replaced by another network or graph if that incorporates time delays or has feedback loops. Such controlled states are referred to as gated state or gated memory, and are part of Long Short Term Memory networks and gated recurrent units. This is also called Feedback Neural Network.

Long Short Term Memory

Long Short Term Memory is an artificial recurrent neural network architecture used in the field of deep learning. Unlike standard feedforward neural networks, Long Short Term Memory has feedback connections. It can process not only single data points (such as images), but also entire sequences of data (such as speech or video). For example, Long Short Term Memory is applicable to tasks such as unsegmented, connected handwriting recognition, speech recognition and anomaly detection in network traffic or intrusion detection systems.

A common Long Short Term Memory unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

Long Short Term Memory networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. Long Short Term Memory units were developed to deal with the vanishing gradient problem that can be encountered when training traditional Recurrent Neural Networks. Relative insensitivity to gap length is an advantage of Long Short Term Memory over Recurrent Neural Networks, hidden Markov models and other sequence learning methods in numerous applications.

Deep Learning

Deep-learning architectures such as deep neural networks, deep belief networks, deep reinforcement learning, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, speech recognition, natural language processing, machine translation, bioinformatics, drug design, medical image analysis, climate science, material inspection and board game programs, where they have produced results comparable to and in some cases surpassing human expert performance.

Artificial neural networks were inspired by information processing and distributed

communication nodes in biological systems. Artificial Neural Networks have various differences from biological brains. Specifically, artificial neural networks tend to be static and symbolic, while the biological brain of most living organisms is dynamic (plastic) and analogue.

The adjective "deep" in deep learning refers to the use of multiple layers in the network. Early work showed that a linear perceptron cannot be a universal classifier, but that a network with a nonpolynomial activation function with one hidden layer of unbounded width can. Deep learning is a modern variation which is concerned with an unbounded number of layers of bounded size, which permits practical application and optimised implementation, while retaining theoretical universality under mild conditions. In deep learning the layers are also permitted to be heterogeneous and to deviate widely from biologically informed connectionist models, for the sake of efficiency, trainability and understandability, whence the "structured" part.

Some GlassDoor Deep Learning Interview Questions tasks:

How to evaluate a Machine Learning model?

https://towardsdatascience.com/various-ways-to-evaluate-a-machine-learning-models-performance-230449055f15

Confusion matrix Accuracy Precision Recall Specificity F1 score Precision Recall curve Receiver Operating Characteristics curve

Support Vector Machines

In machine learning, support-vector machines are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis. Developed at ATT Bell Laboratories by Vladimir Vapnik with colleagues. Support Vector Machines are one of the most robust prediction methods, being based on statistical learning frameworks or Vapnik Chervonenkis theory. Given a set of training examples, each marked as belonging to one of two categories, a Support Vector Machine training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use Support Vector Machines in a probabilistic classification setting). Support Vector Machine maps training examples to points in space so as to maximise the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap

they fall.

In addition to performing linear classification, Support Vector Machines can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

When data are unlabelled, supervised learning is not possible, and an unsupervised learning approach is required, which attempts to find natural clustering of the data to groups, and then map new data to these formed groups. The support-vector clustering algorithm, applies the statistics of support vectors, developed in the support vector machines algorithm, to categorise unlabeled data.

Multi Layer Perceptron

Notes.

What are the typical algorithms to do efficient convolutions?

Convolution. Fast Fourier Transformation. Circular convolution theorem. Graphics Processing Units. Mega Kernels. Winograd Convolution Algorithm. Tensile Stress. Batched General Matrix Matrix Multiplication operation.

Questions on deep learning and tensorflow. Implement a convolutional layer in Python.

I will write up a little file called Deep Learning.py which contains the Kaggle course and then some to skim and review with comments.

If I have an input image of 4000×4000 and objects that I want to detect of size 8×8, what are the limiting factors in an object detector that would drive the performance in detecting the 8×8 objects?

One might think that this is rather large input of 16000000 to convolve upon it would depend upon the objects to be detected as well. Whether or not one could simply execute some algorithm upon a lossy downsized say 2000×2000 scanning for a 4×4 and use that to guide an upstream verificational algorithm is 1 simple little idea rather. Perhaps one that is sort of prone to over capturing slightly and then closely examine those potential target regions for more superior matches in the upstream higher resolution image file.

Explain dropouts and why we use them.

Large weights in a neural network are a sign of a more complex network that has

overfit the training data. Probabilistically dropping out nodes in the network is a simple and effective regularisation method. A large network with more training and the use of a weight constraint are suggested when using dropout.

Describe regularisation techniques. How to compute the number of parameters in a Convolutional Neural Network.

One of the major aspects of training your machine learning model is avoiding overfitting. The model will have a low accuracy if it is overfitting. This happens because your model is trying too hard to capture the noise in your training dataset. By noise we mean the data points that do not really represent the true properties of your data, but random chance. Learning such data points, makes your model more flexible, at the risk of overfitting. The concept of balancing bias and variance, is helpful in understanding the phenomenon of overfitting. L2 Ridge and L1 Lasso recall from Interviews.pdf of course. A standard least squares model tends to have some variance in it, i.e. this model will not generalise well for a dataset different than its training data. Regularisation, significantly reduces the variance of the model, without substantial increase in its bias. So the tuning parameter λ, used in the regularization techniques described above, controls the impact on bias and variance. As the value of λ rises, it reduces the value of coefficients and thus reducing the variance. Till a point, this increase in λ is beneficial as it is only reducing the variance(hence avoiding overfitting), without losing any important properties in the data. But after certain value, the model starts loosing important properties, giving rise to bias in the model and thus underfitting. Therefore, the value of λ should be carefully selected for example using k-fold cross validation.

It is just like some sum layer to layer pairwise over induced connections between those layers.

Basic deep learning questions such as batch normalisation, rectified linear unit, convolution, etc. Open questions related to point cloud.

Batch normalisation is a method used to make artificial neural networks faster and more stable through normalisation of the layers' inputs by re-centering and re-scaling.

While the effect of batch normalisation is evident, the reasons behind its effectiveness remain under discussion. It was believed that it can mitigate the problem of internal covariate shift, where parameter initialisation and changes in

the distribution of the inputs of each layer affect the learning rate of the network. Recently, some scholars have argued that batch normalisation does not reduce internal covariate shift, but rather smooths the objective function, which in turn improves the performance. However, at initialisation, batch normalisation in fact induces severe gradient explosion in deep networks, which is only alleviated by skip connections in residual networks. Others sustain that batch normalisation achieves length-direction decoupling, and thereby accelerates neural networks. More recently a normalise gradient clipping technique and smart hyperparameter tuning has been introduced in Normaliser Free Networks, so called "NF-Nets" which mitigates the need for batch normalisation.

The Point Cloud Library (PCL) is an open-source library of algorithms for point cloud processing tasks and 3D geometry processing, such as occur in three-dimensional computer vision. The library contains algorithms for filtering, feature estimation, surface reconstruction, 3D registration, model fitting, object recognition, and segmentation. Each module is implemented as a smaller library that can be compiled separately. PCL has its own data format for storing point clouds - PCD (Point Cloud Data), but also allows datasets to be loaded and saved in many other formats. It is written in C++ and released under the BSD license.

These algorithms have been used, for example, for perception in robotics to filter outliers from noisy data, stitch 3D point clouds together, segment relevant parts of a scene, extract keypoints and compute descriptors to recognize objects in the world based on their geometric appearance, and create surfaces from point clouds and visualise them.

Normaliser Free Networks.

https://medium.com/analytics-vidhya/paper-explained-normalizer-free-nets-nfnets-high-performance-large-scale-image-recognition-75518978b1fe

How do you deal with class imbalance?

One of the most common techniques is to randomly select a subset of the dataset such that the training set is class balanced and then hope to identify the useful predictors. This can come into depending upon penalty functions of course another technique may be to instead harshly penalise that sort of misclassification but this could cause problems depending upon what precisely it is that one wishes to optimise. Collect more data, change performance metric, resampling dataset, generate synthetic samples, different algorithms, penalised models, different

perspective, getting creative.

Implement a K means algorithm.

OK

What happens if you remove non linearities in a neural network?

So I would think the task statement is referring to the construction of the structure of a neural network at which point it would probably degenerate into a simple linear regression or classifier model.

What if you have 2 neural networks: one whose weight matrix is shorter and wider and one whose weight matrix is longer but not wide. Which is better?

What a task statement. Computation. Longer might lead to more vanishing or exploding gradient problem issues. Structures, task details, yadda yadda, both are capable of executing and representing some functions. I think that an interviewer may want one to say that longer deeper networks can be much better from an optimised perspective in a trading firm's codebase production.

Formulas for cross entropy and triplet loss. How does the margin in triplet loss affect training?

$H(p, q) = -E_p[\log(q)]$
$H(p, q) = -\sum_{x \in X} p(x) \log(q(x))$
$H(p, q) = -\int_X P(x) \log(Q(x)) dr(x)$

Triplet loss is a loss function for machine learning algorithms where a reference input (called anchor) is compared to a matching input (called positive) and a non-matching input (called negative). The distance from the anchor to the positive is minimized, and the distance from the anchor to the negative input is maximized. An early formulation equivalent to triplet loss was introduced (without the idea of using anchors) for metric learning from relative comparisons.

By enforcing the order of distances, triplet loss models embed in the way that a pair of samples with same labels are smaller in distance than those with different labels. Unlike t-SNE which preserves embedding orders via probability distributions, triplet loss works directly on embedded distances. Therefore, in its common implementation, it needs soft margin treatment with a slack variable α in its hinge loss-style formulation. It is often used for learning similarity for the purpose of learning embeddings, such as learning to rank, word embeddings,

thought vectors, and metric learning.

Consider the task of training a neural network to recognize faces (e.g. for admission to a high security zone). A classifier trained to classify an instance would have to be retrained every time a new person is added to the face database. This can be avoided by posing the problem as a similarity learning problem instead of a classification problem. Here the network is trained (using a contrastive loss) to output a distance which is small if the image belongs to a known person and large if the image belongs to an unknown person. However, if we want to output the closest images to a given image, we would like to learn a ranking and not just a similarity. A triplet loss is used in this case.

The loss function can be described by means of the Euclidean distance function

$$L(A, P, N) = \max(||f(A) - f(P)||^2 - ||f(A) - f(N)||^2 + \alpha, 0)$$

where A is an anchor input, P is a positive input of the same class as A, N is a negative input of a different class from A, α is a margin between positive and negative pairs, and f is an embedding. This can then be used in a cost function, that is the sum of all losses, which can then be used for minimisation of the posed optimisation problem

$$T = \sum_{i=1}^{M} L(A^{(i)}, P^{(i)}, N^{(i)})$$

The indices are for individual input vectors given as a triplet. The triplet is formed by drawing an anchor input, a positive input that describes the same entity as the anchor entity, and a negative input that does not describe the same entity as the anchor entity. These inputs are then run through the network, and the outputs are used in the loss function.

Describe the difference between a traditional Convolutional Neural Network, Residual Neural Network and Inception network.

In deep learning, a convolutional neural network is a class of artificial neural network, most commonly applied to analyze visual imagery. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on the shared-weight architecture of the convolution kernels or filters that slide along input features and provide translation equivariant responses known as feature maps. Counter-intuitively, most convolutional neural networks are only equivariant, as opposed to invariant, to translation. They have applications in image and video recognition, recommender systems, image classification, image

segmentation, medical image analysis, natural language processing, brain-computer interfaces, and financial time series.

Convolutional Neural Networks are regularised versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "full connectivity" of these networks make them prone to overfitting data. Typical ways of regularisation, or preventing overfitting, include: penalizing parameters during training (such as weight decay) or trimming connectivity (skipped connections, dropout, etc.) Convolutional Neural Networks take a different approach towards regularisation: they take advantage of the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in their filters. Therefore, on a scale of connectivity and complexity, Convolutional Neural Networks are on the lower extreme.

A residual neural network is an artificial neural network. Residual neural networks utilise skip connections, or shortcuts to jump over some layers. Typical residual neural network models are implemented with double- or triple- layer skips that contain nonlinearities [rectified linear unit] and batch normalisation in between. An additional weight matrix may be used to learn the skip weights; these models are known as Highway Networks. Models with several parallel skips are referred to as Dense Networks. In the context of residual neural networks, a non-residual network may be described as a plain network.

There are two main reasons to add skip connections: to avoid the problem of vanishing gradients, or to mitigate the degradation (accuracy saturation) problem; where adding more layers to a suitably deep model leads to higher training error. During training, the weights adapt to mute the upstream layer, and amplify the previously-skipped layer. In the simplest case, only the weights for the adjacent layer's connection are adapted, with no explicit weights for the upstream layer. This works best when a single nonlinear layer is stepped over, or when the intermediate layers are all linear. If not, then an explicit weight matrix should be learned for the skipped connection (a Highway Network should be used).

Skipping effectively simplifies the network, using fewer layers in the initial training stages. This speeds learning by reducing the impact of vanishing gradients, as there are fewer layers to propagate through. The network then gradually restores the skipped layers as it learns the feature space. Towards the end of training, when all layers are expanded, it stays closer to the manifold and thus learns faster. A neural network without residual parts explores more of the

feature space. This makes it more vulnerable to perturbations that cause it to leave the manifold, and necessitates extra training data to recover.

In machine learning, a highway network is an approach to optimising networks and increasing their depth. Highway networks use learned gating mechanisms to regulate information flow, inspired by Long Short Term Memory recurrent neural networks. The gating mechanisms allow neural networks to have paths for information to follow across different layers ("information highways").

Highway networks have been used as part of text sequence labeling and speech recognition tasks.

https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

Given this tensor, how would you normalise it, using 32 cores? Is mutex a good idea for this?

Does not seem too critical, one can Google literature. I do not know, one supposes that this sort of a task is all very well studied and canonical and if a firm wants one to examine they can task one with that upon which one can.

Compare different activation functions.

Binary Step

$f(x) = 1$ for $x \geq 0$ and $f(x) = 0$ for $x < 0$

Binary classifier. Simple.

Linear

$f(x) = ax$

Problem gradient is the same for every iteration so the network will not be able to train and capture the complex patterns from the data. Might be useful for simple tasks where interpretability is highly desired.

Sigmoid

$f(x) = \frac{1}{1+e^{-x}}$

Transforms the values into the range $[0, 1]$ can represent probabilities. Continuous and differentiable at all points desiderata.

Tanh

$\tanh(x) = \frac{2}{1+e^{-2x}} - 1$

Scaled sigmoid into $[-1, 1]$. Sometimes preferred over the sigmoid function since it is 0 centered and the gradients are not restricted to move in a certain direction.

Rectified Linear Unit

$f(x) = \max(0, x)$

$f(x) = x$ for $x \geq 0$ and $f(x) = 0$ for $x < 0$

Does not activate all the neurons at the same time can be useful. The neurons will only be deactivated if the output of the linear transformation is less than 0. This 0 derivative on $x < 0$ can cause the weights and biases for some neurons to not be updated which can create never activated dead neurons.

Leaky Rectified Linear Unit

$f(x) = x$ for $x \geq 0$ and $f(x) = ax$ for $x < 0$ e.g. $\frac{1}{100} \cdot x$

Parameterised Rectified Linear Unit

As before, a is a trainable parameter. The network also learns the value of a for faster and more optimal convergence.

Exponential Linear Unit

$f(x) = x$ for $x \geq 0$ and $f(x) = a\left(e^x - 1\right)$ for $x < 0$

Uses a logarithmic curve for defining the negative values.

Swish

$f(x) = \frac{x}{1-e^{-x}}$

Used by researchers at the Google firm. Differentiable at all points. Not monotonic. Can outperform rectified linear unit.

Softmax

$f(x)_j = \frac{e^{z_j}}{\sum e^{z_i}}$

Most often used as a final layer in multiclass classification tasks.

What is Principal Component Analysis?

The principal components of a collection of points in a real coordinate space are a sequence of p unit vectors, where the ith vector is the direction of a line that best fits the data while being orthogonal to the first $i-1$ vectors. Here, a best-fitting line is defined as one that minimises the average squared distance from the points to the line. These directions constitute an orthonormal basis in which different individual dimensions of the data are linearly uncorrelated. Principal component analysis (PCA) is the process of computing the principal components and using them to perform a change of basis on the data, sometimes using only the first few principal components and ignoring the rest.

Principal Component Analysis is used in exploratory data analysis and for making predictive models. It is commonly used for dimensionality reduction by projecting each data point onto only the first few principal components to obtain lower-dimensional data while preserving as much of the data's variation as possible. The first principal component can equivalently be defined as a direction that maximises the variance of the projected data. The ith principal component can be taken as a direction orthogonal to the first $i-1$ principal components that maximises the variance of the projected data.

For either objective, it can be shown that the principal components are eigenvectors of the data's covariance matrix. Thus, the principal components are often computed by eigendecomposition of the data covariance matrix or singular value decomposition of the data matrix. Principal Component Analysis is the simplest of the true eigenvector-based multivariate analyses and is closely related to factor analysis. Factor analysis typically incorporates more domain specific assumptions about the underlying structure and solves eigenvectors of a slightly different matrix. Principal Component Analysis is also related to canonical correlation analysis (CCA). CCA defines coordinate systems that optimally describe the cross-covariance between two datasets while Principal Component Analysis defines a new orthogonal coordinate system that optimally describes variance in a single dataset. Robust and L1-norm-based variants of standard Principal Component Analysis have also been proposed.

What is the Fourier transform of a Kronecker delta function and what's the intuition behind the Fourier transform of a Gaussian?

I forget sometimes and on my radar at some point I need to take all of this Fourier stuff very seriously and then all of the Ergodic stuff seriously I want to be

a sharp whizz young lad like Benjamin Krause and know the things and stuff too that they write down in the smart papers of mathematiques like him he is a hero and inspiration midkey! So a Kronecker delta function is the most simple indicator function extension can be like a discrete rectangle generalised dimensions whatever and there exists something called the Scaling Theorem, some sinc rect functions, approximations to a Kronecker delta function based upon linear multiples trigonometric functions which confer useful signal representational and computational properties.

Why use the softmax layer? Why use an activation function?

A final soxtmax layer also known as softargmax or normalised exponential function is a generalisation of the logistic function to multiple dimensions. Recall an activation function is just one of the most simple core components in the structure of an artificial neural network.

What is the definition of cross entropy?

In information theory, the cross-entropy between 2 probability distributions p and q over the same underlying set of events measures the average number of bits needed to identify an event drawn from the set if a coding scheme used for the set is optimised for an estimated probability distribution q, rather than the true distribution p.

If I have labeled images where I want to perform object detection, and sometimes the objects are partially occluded, how can I exclude partially occluded objects from training?

There exists literature. To remove partial occlusion from a foreground object, the vision problem is to determine the boundary of the completely occluded region [green curve] and the width of the partially occluded region [the length of the red arrow].

What is the difference between Region Based Convolutional Neural Networks, Fast Region Based Convolutional Neural Network, and Faster Region Based Convolutional Neural Network?

Computer vision, object detection. Take an input image and produce a set of bounding boxes as output, where each bounding box contains an object and also the category of the object. Something called Region Of Interest Pooling and reshaping and then integration of the Region Of Interest generation into the

neural network itself.

Maximum A Posteriori and predictive distribution, how does K Nearest Neighbours work and tradeoffs?

So now I ideate a little about tests, algorithms, and pre processing observations that could be made upon a dataset which could indicate that running a k nearest neighbours algorithm may be unwise, suboptimal, a bad choice. Do they exist? What can be said in simple statistical terms? Classification or regression. Can further involve not just weighting all of the k nearest neighbours equally but rather giving each neighbour a weight of $\frac{1}{\text{distance}}$.

Difference between Adaptive Moment Estimation and Stochastic Gradient Descent optimiser.

Optimisation technique for gradient descent. It requires less memory and is efficient. Intuitively, it is a combination of the gradient descent with momentum algorithm and the Root Mean Square Propagation algorithm. Momentum is used to accelerate the gradient descent algorithm by taking into consideration the exponentially weighted average of the gradients. Using averages makes the algorithm converge towards the minima in a faster pace. So Root mean square propagation is an adaptive learning algorithm that tries to improve adaptive gradient. Instead of taking the cumulative sum of squared gradients it takes the exponential moving average. So usually a standard batch gradient descent method would perform an iteration $w := w - m\nabla Q(w) = w - \frac{m}{n}\sum_{i=1}^{n} \nabla Q_i(w)$ with a step size learning rate parameter and the summand gradient functions can require expensive evaluations of the gradients. To economise on the computational cost at every iteration, stochastic gradient descent samples a subset of summand functions at every step. In stochastic or online gradient descent the true gradient is approximated by a gradient at a single sample. Some potential benefits of Adaptive Moment Estimation on non convex optimisation problems are: straightforward to implement, computationally efficient, little memory requirements, invariant to diagonal rescale of the gradients, well suited for problems that are large in terms of data and/or parameters, appropriate for non stationary objectives, appropriate for problems with very noisy/or sparse gradients, hyper parameters have intuitive interpretation and typically require little tuning.

In random forest, what is random? The sample points or the features?

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned. Random decision forests correct for decision trees' habit of overfitting to their training set. Random forests generally outperform decision trees, but their accuracy is lower than gradient boosted trees. However, data characteristics can affect their performance.

So the training algorithm for most random forest algorithms is the general technique of boostrap aggregating to tree learners. It repeatedly selects a random sample with replacement from the training dataset and fits trees to these samples.

However random forests can also refer to another type of scheme: a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. This process is also known as feature bootstrap aggregating.

Efficient ways to parallelise matrix multiplication.

Tutorials Point has a piece titled Parallel Algorithm - Matrix Multiplication. There exists literature. Mesh network. Hypercube network. Block matrix decomposition.

What causes gradient vanishing? What is overfitting? What is rectified linear unit and leaky rectified linear unit? What is the component of a Long Short Term Memory network?

As more layers using certain activation functions are added to neural networks, the gradients of the loss function approaches 0, making the network hard to train. Certain activation functions, like the sigmoid function, squish a large input space into a small input space. Therefore, a large change in the input of the sigmoid function will cause a small change in the output. Hence, the derivative becomes small. The simplest solution is to use other activation functions such as rectified linear units. Overfitting of course is when one produces a model which is too tightly fit to a training set, and underperforms on a test set. Too much noise has been captured rather than the optimal capture of signal. Residual networks are another solution, as they provide residual connections straight to earlier larger layers. This residual connection does not go through activation functions that squish the derivatives, resulting in a higher overall derivative of the block.

Finally, batch normalisation layers can also resolve the issue. Batch normalisation reduces this problem by simply normalising the input so that $|x|$ does not reach the outer edges of the sigmoid function. It normalises the input so that most of it falls in the region where the derivative is not too small. A leaky rectified linear unit is one where the function is not 0 on $x < 0$, rather there has been some leakage and a function signal still goes through but it has smaller derivative gradient e.g. $f(x) = x$ on $x \geq 0$ and $f(x) = \frac{1}{100} \cdot x$ on $x < 0$. And such a coefficient parameter can be learned, optimised in an algorithm, this is parametric rectified linear unit. And there exist other variants in the literature which could be enumerated at length but a brief skim suffices once in awhile to be an expert tier whizzy whizz. So from lectures circa the middle segment of the Carnegie Mellon University course and elsewhere, the primary component of a Long Short Term Memory network is known as a cell. Which can process data sequentially and keep its hidden state through time. It is a rather nontrivial construction one can lowkey comprehend as such at a glance, but there exists a notion of a forget gate, peephole, convolutional peephole. Well rather a cell, an input gate, an output gate, and a forget gate. The cell remembers values over arbitrary time intervals. Well suited to classifying, processing, and making predictions based on time series data.

What happens if you set all the weights to 0 in a neural network with back propagation?

If new weights are multiples of old weights, then 0 weights can never be changed. In such an algorithm they are a constant invariant 0 under all operation. When back propagation involves adding to the existing weights, not multiplying. Then the amount added is specified by the delta rule. First, neural networks tend to get stuck in local minima, so it is a good idea to give them many different starting values. Second, if the neurons start with the same weights, then all of the neurons will follow the same gradient, and will always end up doing the same thing as one another. If weights are thought of as priors, then one has ruled out any possibility that thee inputs could possibly affect the system. How could any gradient descent algorithm be oriented in terms of determining the direction of the system? One is placing oneself on a saddle point of the parameter space. If one has sigmoid activation or anything where $g(0) \neq 0$ then it will cause weights to move together, limiting the power of back propagation to search the entire space. If you have tanh or Rectified Linear Unit acitvation or anything where $g(0) = 0$ then all of the outputs will be 0, and the gradients for the weights will always be 0. Hence one will not have any learning at all. One can find some

simple commentary on how to go about selecting weights here:

https://staff.itee.uq.edu.au/janetw/cmc/chapters/BackProp/index2.html

What is Generative Adversarial Network, Variational Autoencoder?

A Generative Adversarial Network is a class of machine learning frameworks designed by Ian Goodfellow and his colleagues in June 2014. Two neural networks contest with each other in a game (in the form of a 0 sum game, where one agent's gain is another agent's loss).

Given a training set, this technique learns to generate new data with the same statistics as the training set. For example, a Generative Adversarial Network trained on photographs can generate new photographs that look at least superficially authentic to human observers, having many realistic characteristics. Though originally proposed as a form of generative model for unsupervised learning, Generative Adversarial Networks have also proved useful for semi-supervised learning, fully supervised learning, and reinforcement learning.

The core idea of a Generative Adversarial Network is based on the "indirect" training through the discriminator, another neural network that is able to tell how much an input is "realistic", which itself is also being updated dynamically. This basically means that the generator is not trained to minimise the distance to a specific image, but rather to fool the discriminator. This enables the model to learn in an unsupervised manner.

Fine tune a Visual Geometry Group classifier for some task.

Recall the standard usual multi layer convolutions. And scanning lined up scans so that one obtains shift invariance. And the key idea to also introduce distorted versions of the inputs as additional input data points one can add noise to them as training data.

How To Use A Pre-Trained Model [Visual Geometry Group] For Image Classification

https://towardsdatascience.com/how-to-use-a-pre-trained-model-vgg-for-image-classification-8dd7c4a4a517

Kaggle Using Pre Trained Visual Geometry Group Model

Describe the working of a Long Short Term Memory Model.

In theory, classic (or "vanilla") Recurrent Neural Networks can keep track of arbitrary long-term dependencies in the input sequences. The problem with vanilla Recurrent Neural Networks is computational (or practical) in nature: when training a vanilla Recurrent Neural Network using back-propagation, the long-term gradients which are back-propagated can "vanish" (that is, they can tend to zero) or "explode" (that is, they can tend to infinity), because of the computations involved in the process, which use finite-precision numbers. Recurrent Neural Network using Long Short Term Memory units partially solve the vanishing gradient problem, because Long Short Term Memory units allow gradients to also flow unchanged. However, Long Short Term Memory networks can still suffer from the exploding gradient problem.

Implement a Compute Unified Device Architecture [CUDA] kernel that computes the frequencies of numbers given in an array - Given a very large file containing integers, implement an algorithm that outputs sorted version. (also do space and time complexity analysis) - Implement an optimised Compute Unified Device Architecture [CUDA] kernel for matrix transpose.

The Github reposistory "learn-cuda" contains much content to lightly trawl and comprehend some tasks which can be made more efficient with Compute Unified Device Architecture [CUDA] implementations.

https://github.com/kevinzakka/learn-cuda/blob/master/src/histogram.cu

Light Gradient Boosting Machines.

Think about Adaptive Moment Estimation. Adaptive Boosting. Takes less time to train than Extreme Gradient Boosting. Recall like ensemble learning can be rather than training 1 tree on an entire dataset which can also have computational implications. That instead one can randomly split the dataset, train multiple trees, and then a simple vote can produce superior performance. The key idea might be that boosting algorithms reduce bias and variance more effectively than the boostrap aggregation techniques which are more prone to decreasing variance. So boosting has higher weights on misclassified data points. Ensemble model. Tree specific parameters. Boosting parameters. Miscellaneous parameters. So Extreme Gradient Boosting has level wise tree growth whereas Light Gradient Boosting Machines have leaf wise tree growth. Faster training speed and higher efficiency. Lower memory usage. Better accuracy than any other boosting algorithm. Compatibility with large datasets. Parallel learning supported.

Extreme Gradient Boosting.

Uh huh Lex Fridman likes to tell kiddos that what is interesting to him is the task of being a human and spotting instances where one can apply these easy powerful techniques. This task statement says the Human Resource departments woke up to the utility of machine learning. So I must begin to always be sharp and on the lookout for moments where data can be captured or already exists that could be useful. Regularisation. Parallel processing. Making a tree using all cores. Each tree can be built only after the previous one. High flexibility. Custom optimisation objectives and evaluation criteria. Handling missing values. A gradient boosting machine may be seen as a greedy algorithm, halting upon a split of negative loss -2 whereas Extreme Gradient Boosting makes the splits and prunes backwards, removing those beyond which there is no positive gain. Thus it will capture if this -2 split is followed by a +10 split and outperform. Build in cross validation at each iteration of the boosting process.